1. MapReduce实现基本SQL操作的原理

```
in  exists      在 mapper 阶段，写一个if 过滤
order by        每个reduceTask 排序
sort by
having          group by --> select ---> having reducer阶段
limit           reducer阶段

case  when ... then ... when ..  then .. else ... end    mapper阶段实现，简单的if
else 就能做
```

1.1. join 实现

代码：SQL ====> 该怎么写MapReduce?（MapReduce Join实现： ReduceJoin MapJoin)

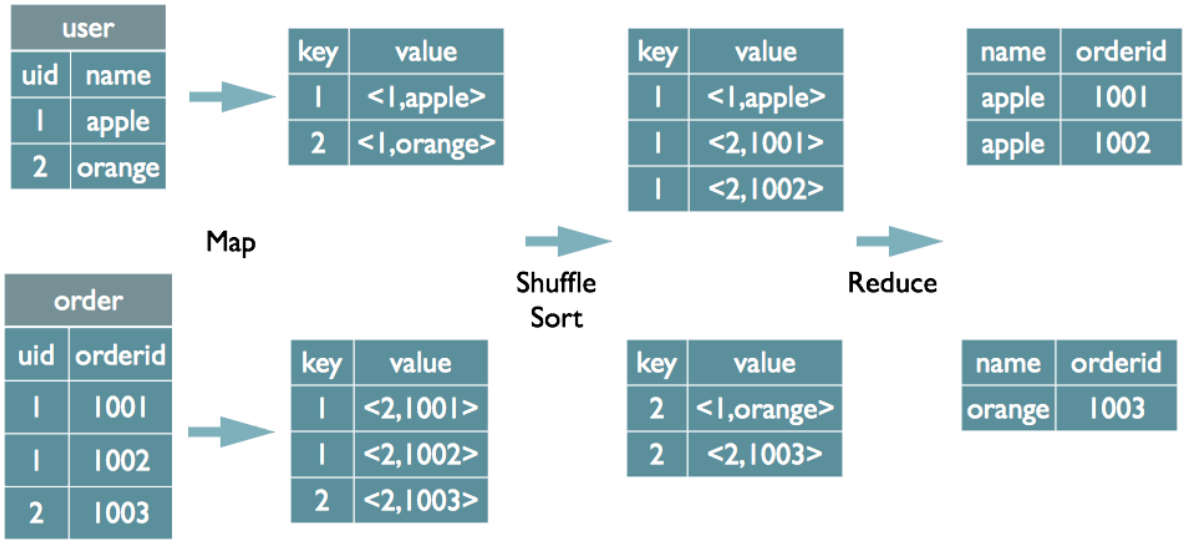
```
select u.name, o.orderid from order o join user u on o.uid = u.uid;
```

```
mapper: 映射（提取value, 提取key来标识这个value）
shuffle: 把所有的mapTask提取的key-value对进行混洗，把相同的key的所有的value汇聚到一起
reduce: 每次拿出来一个key的所有value做一次聚合操作
```

思路实现：

```
1、思考这个SQL要使用几个MR来实现
2、每个MR的mapper的输出key-value是什么
```

实现思路图：



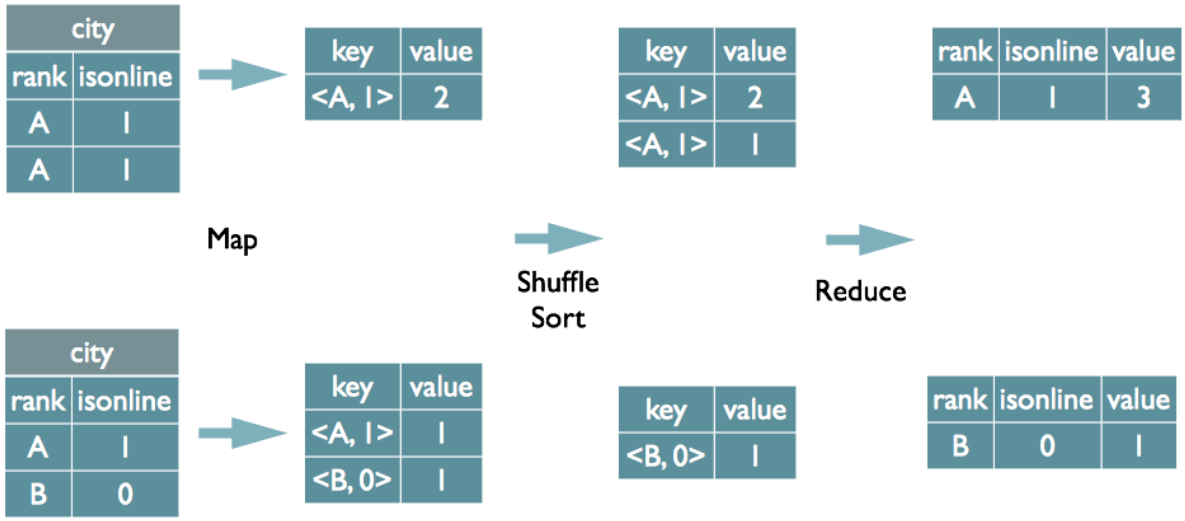
1.2. group by 实现

代码：既然MR一定会按照key进行分区和排序，最后还会进行分组，那么既然你的 sql 包含 group by 这个分组逻辑，所以必然你的 分区字段中，要包含：这个SQL 的分组字段

```
select rank, isonline, count(*) from city group by rank, isonline;
```

```
mapreduce的mapper的key： rank, isonline
mapreduce的reducer的逻辑： 统计每一组的元素的个数
```

实现思路图：



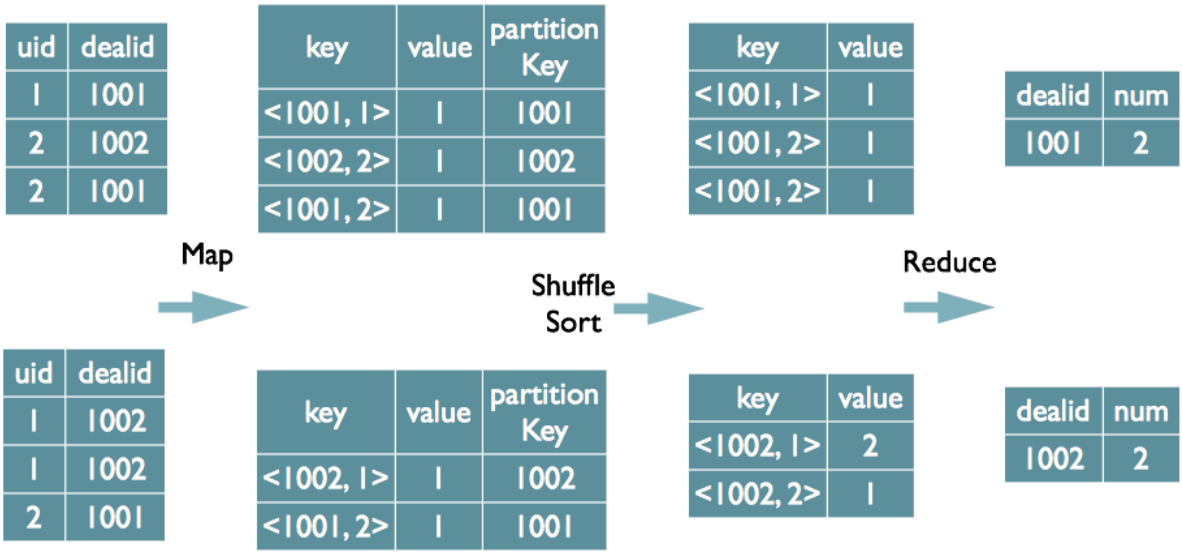
1.3. distinct 实现

代码：

```
-- 直接按照 uid 分组即可
select count(distinct uid) num from order;

-- 按照 dealid 和 uid 联合分组
select dealid, count(distinct uid) num from order group by dealid;
```

实现思路图：



1.4. 企业面试案例

求共同好友

```
A：B,C,D,E
B：C,D
C：A,E
D：B,C
E：A,B
```

结果：

```
A-B：C,D
A-C：E
A-D：B,C
```

倒推：

```
A-B：C
A-B：D
```

倒推：C：A-B

```
C-A
C-B
```

再倒过来：

```
A-C
B-C
```

原始数据的解析：

```
A：B,C,D,E
A-B
A-C
A-D
A-E
```

2. Hive的HQL怎么转换成MapReduce?

了解了MapReduce实现SQL基本操作之后，我们来看看Hive是如何将SQL转化为MapReduce任务的，整个编译过程分为六个阶段：

```
1、Antlr定义SQL的语法规则，完成SQL词法，语法解析，将SQL转化为抽象语法树AST Tree
2、遍历AST Tree，抽象出查询的基本组成单元QueryBlock -> QB  理解成 子查询 最小的查询执行单元
3、遍历QueryBlock，翻译为执行操作树OperatorTree  Operator不可拆分的一个逻辑执行单元
4、逻辑层优化器进行OperatorTree变换，合并不必要的ReduceSinkOperator，减少shuffle数据量
5、遍历OperatorTree，翻译为MapReduce任务,将逻辑执行计划转换成物理执行计划TaskTree
6、物理层优化器进行MapReduce任务的变换，生成最终的执行计划
```

```
Hive的四大组件：Driver ----> Compiler  ---->  Optimizer ---> Executor
```

```
Driver: CliDriver  Diver ParseDriver ===>  ASTNode
Compiler:  ASTNode ===> OperatorTree ===>  TaskTree
Optimizer:  OperatorTree ===>  TaskTree
Executor: Driver.executTask() : 提交任务到hadoop集群运行
```

```
Driver
Compiler
Optmizer
Executor
```

3. Hive源码的入口

注意关于阅读源码的几个细节：

```
1、版本的选择和确定
    hive-1.x hive-2.x hive-3.x    不新不旧的稳定版本
2、阅读源码的项目的搭建
3、编译 调试运行
    windows平台编译比较困难
4、源码阅读的入口
    场景驱动
```

```
    1、HDFS
        集群启动 namenode  datanode  secondarynamenode
        上传下载  put   get
    2、zookeeper
        集群启动
        leader选举
        读写请求
    3、mapreduce
        一个job的提交和执行的完整的流程
      spark
        一个spark的application的完整的提交和执行
    4、hive
        SQL ----> mapreduce
        你在哪里个地方提交SQL?
        1、hive> select .....
        2、hive -e | hive -f ....
        3、JDBC程序
        4、Hiveserver2 + beeline
```

hive ---> sql

hive -e "sql" hive -f "hive.file"

```
Hive的java代码的执行入口：CliDriver
java .....CliDriver -Dxxkey==xxvalue -Dxxkey==xxvalue
```

所以，最终确定 入口就是： CliDriver.main()

```
怎么构建看源码的环境：
1、从官网下载对应版本的源码包
2、直接解压缩到某个目录。
3、通过IDEA直接打开这个项目即可
```

```
不编译最大的区别就是：HiveParser.java    hiveparser.g 词法文件解析的工作
SQL --->  ASTTree
```

4. 详细过程

4.1. 第一阶段：SQL词法，语法解析

4.1.1. antlr介绍

Hive 使用 Antlr 实现 SQL 的词法和语法解析。Antlr 是一种语言识别的工具，可以用来构造领域语言。**Antlr 完成了词法分析、语法分析、语义分析、中间代码生成的过程。**

Antlr的工作方式：编写一个语法文件，构造特定规则的语法，定义定义语法和词法规则完成最终的替换，生成代码。

Hive的语法规则和词法规则，都是定义在类似于 xxx.g 的文件中。其中：

0.10x版本以前：

```
一个统一的语法和词法文件：Hive.g
```

0.11版本往后：

```
定义词法规则：HiveLexer.g
定义语法规则：SelectClauseParser.g，FromClauseParser.g，IdentifiersParser.g，
HiveParser.g
```

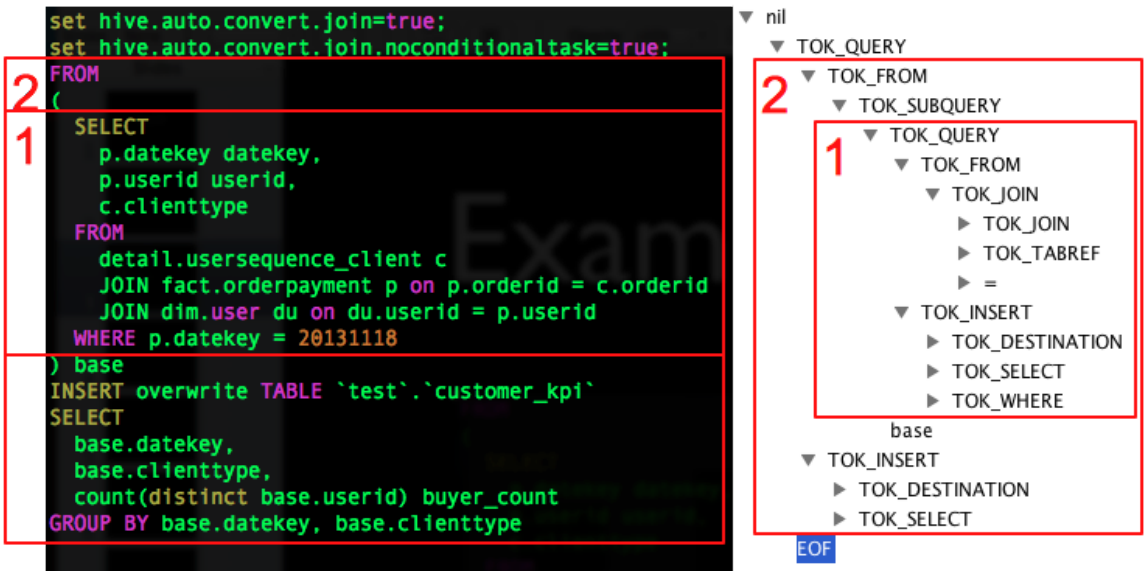
Hive 接收到用户编写的 HQL，就通过 antlr 进行解析生成代码。

4.1.2. 第一阶段：SQL生成抽象语法树AST Tree

Antlr 对 Hive SQL 解析的代码如下，HiveLexerX，HiveParser 分别是 Antlr 对语法文件 Hive.g 编译后自动生成的词法解析和语法解析类，在这两个类中进行复杂的解析。

代码跳转关系：

```
CliDriver.main()
CliDriver.run()
CliDriver.executeDriver()
CliDriver.processLine()
CliDriver.processCmd()
CliDriver.processLocalCmd()          # 完整的执行，输出过程
Driver.run()
Driver.runInternal()                 # 编译和执行
Driver.compileInternal()             # 编译
Driver.compile()              # 编译：SQL -> AST -> ResovleTree -> OperatorTree ->
TaskTree
ParseUtils.parse()
ParseDriver.parse()
```



4.1.3. 第二阶段：SQL基本组成单元QueryBlock

AST Tree 仍然非常复杂，不够结构化，不方便直接翻译为MapReduce程序，AST Tree 转化为 QueryBlock 就是将SQL 进一部抽象和结构化。

QueryBlock 是一条 SQL 最基本的组成单元，包括三个部分：输入源，计算过程，输出。QueryBlock 可理解子查询

详见源代码：

```
org.apache.hadoop.hive.ql.parse.QB
```

AST Tree 生成 QueryBlock 的过程是一个递归的过程，先序遍历 AST Tree，遇到不同的 Token 节点，保存到相应的属性中，主要包含以下几个过程：

```
TOK_QUERY => 创建QB对象，循环递归子节点
TOK_FROM => 将表名语法部分保存到QB对象的aliasToTabs等属性中
TOK_INSERT => 循环递归子节点
TOK_DESTINATION => 将输出目标的语法部分保存在QBParseInfo对象的nameToDest属性中
TOK_SELECT => 分别将查询表达式的语法部分保存在destToSelExpr、destToAggregationExprs、
destToDistinctFuncExprs三个属性中
TOK_WHERE => 将where部分的语法保存在QBParseInfo对象的destToWhereExpr属性中
```

4.1.4. 第三阶段：逻辑操作符Operator

Hive 最终生成的 MapReduce 任务，Map 阶段和 Reduce 阶段均由 OperatorTree 组成。逻辑操作符，就是在Map 阶段或者 Reduce 阶段完成单一特定的操作。

基本的操作符包括 TableScanOperator，SelectOperator，FilterOperator，JoinOperator、GroupByOperator，ReduceSinkOperator

由于Join/GroupBy/OrderBy均需要在Reduce阶段完成，所以在生成相应操作的Operator之前都会先生成一个ReduceSinkOperator，将字段组合并序列化为Reduce Key/value, Partition Key

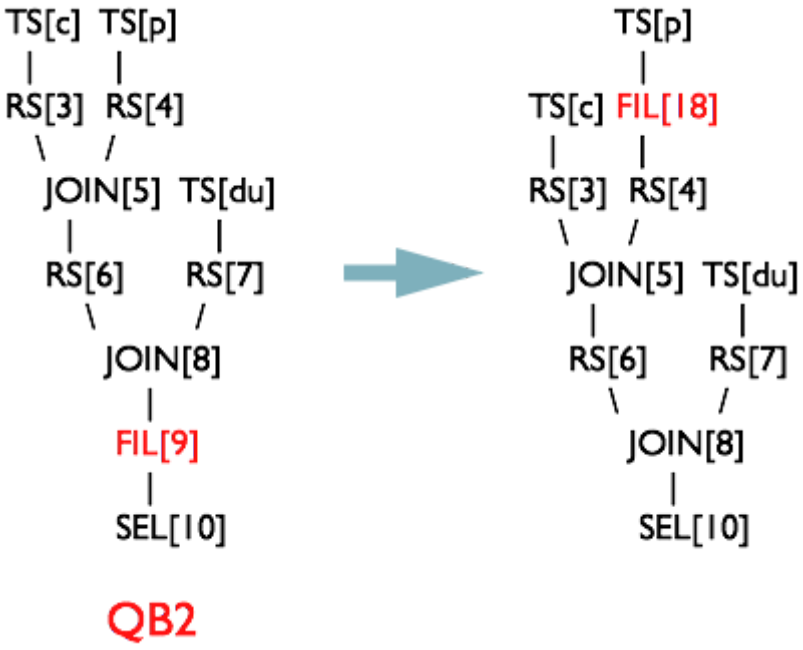
4.1.5. 第四阶段：逻辑层优化器

大部分逻辑层优化器通过变换OperatorTree，合并操作符，达到减少MapReduce Job，减少shuffle数据量的目的。

名称	作用
② SimpleFetchOptimizer	优化没有GroupBy表达式的聚合查询
② MapJoinProcessor	MapJoin，需要SQL中提供hint，0.11版本已不用
② BucketMapJoinOptimizer	BucketMapJoin
② GroupByOptimizer	Map端聚合
① ReduceSinkDeDuplication	合并线性的OperatorTree中partition/sort key相同的reduce
① PredicatePushDown	谓词前置/谓词下推
① CorrelationOptimizer	利用查询中的相关性，合并有相关性的Job，HIVE-2206
② ColumnPruner	字段剪枝

表格中①的优化器均是一个Job干尽可能多的事情/合并。②的都是减少shuffle数据量，甚至不做Reduce。

PredicatePushDown优化器：



SQL 示例：谓词下推：有些动作能先执行的话，就尽量先执行。

```
select a.*, b.* from a join b on a.id = b.id where a.id > 18;

select a.*, c.* from a join (select b.* from b where b.id > 18) c on a.id =
c.id;
```

4.1.6. 第五阶段：OperatorTree生成MapReduce Job的过程

OperatorTree 转化为 MapReduce Job 的过程分为下面几个阶段：

```
1、对输出表生成MoveTask
2、从OperatorTree的其中一个根节点向下深度优先遍历
3、ReduceSinkOperator标示Map/Reduce的界限，多个Job间的界限
4、遍历其他根节点，遇过碰到JoinOperator合并MapReduceTask
5、生成StatTask更新元数据
6、剪断Map与Reduce间的Operator的关系
```

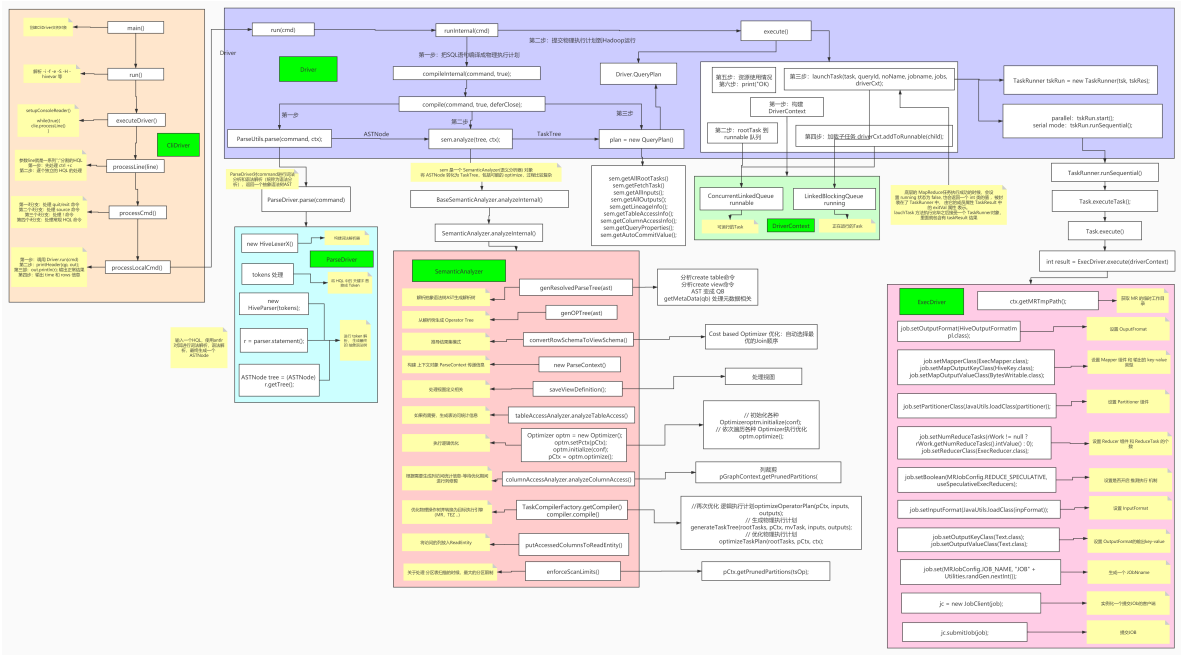
4.1.7. 第六阶段：物理层优化器

这里不详细介绍每个优化器的原理，单独介绍一下MapJoin的优化器

名称	作用
Vectorizer	HIVE-4160，在0.13中发布
SortMergeJoinResolver	与bucket配合，类似于归并排序
SamplingOptimizer	并行order by优化器，在0.12中发布
CommonJoinResolver + MapJoinResolver	MapJoin优化器

5. Hive的HQL编译源码解读

大图：https://blog.csdn.net/zhongqi2513/article/details/107153698



6. Hive SQL编译过程的设计

从上述整个SQL编译的过程，可以看出编译过程的设计有几个优点值得学习和借鉴

1. 使用Antlr开源软件定义语法规则，大大简化了词法和语法的编译解析过程，仅仅需要维护一份语法文件即可。

2. 整体思路很清晰，分阶段的设计使整个编译过程代码容易维护，使得后续各种优化器方便的以可插拔的方式开关，譬如Hive 0.13最新的特性Vectorization和对Tez引擎的支持都是可插拔的。

3. 每个Operator只完成单一的功能，简化了整个MapReduce程序。